Remarks/Arguments

The preceding preliminary amendments and following remarks are submitted in response to the Final Office Action of the Examiner mailed May 31, 2005, setting a three month shortened statutory period for response ending August 31, 2005. Claims 1-8 and 10-25 remain pending, with claim 25 being newly presented. Claim 9 has been canceled without prejudice. Reconsideration, examination and allowance of all pending claims are respectfully requested.

On page 2 of the Final Office Action, the Examiner rejected claims 1-10 under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,125,364 to Greef et al. After careful review, Applicant must respectfully disagree.

Claim 1 recites:

> 1. (Original) A method for creating a plurality of objects from data in persistent storage, the objects having object pointers, unique object identifiers, and object types as attributes, the method comprising the steps of:
> reading the total number of type sets;
> for each type set, reading the total number of objects in each type set;
> for each object in said type set, creating an object from said data in persistent storage;
> for each object pointer in said objects, obtaining the unique object identifier corresponding to said object pointer; and
> for each obtained unique object identifier, obtaining the object address corresponding to said unique object identifier and setting each of said object pointers to said corresponding object addresses.

As can be seen, claim 1 is a method claim and recites a number of method steps. Claim 1 recites the steps of "reading the total number of type sets", and "for each type set, reading the total number of objects in each type set". On page 3 of the Final Office Action, the Examiner states that the first step is disclosed by item 401 of Figure 10 of Greef et al., and the second step is

disclosed by item 402 of Figure 10 of Greef et al. The Examiner states that the offset represents the total number of type sets.

After careful review, Applicant must respectfully disagree. Figure 10 is a flow chart showing how the framework creates an object (i.e. single object) in the volatile storage. (see, column 6, lines 42-44). Figure 10 is a direct sequence from FIG. 9 at 304 "B". Figure 9 starts at 300, and after the identification of the object (i.e. single object) to be retrieved from non-volatile storage is obtained. As such, it is difficult to see how Figure 10 can possibly teach a "method for creating a plurality of objects from data in persistent storage", and more specifically, how item 402 of Figure 10 can teach "for each type set, reading the total number of objects in each type set". It is also difficult to see how item 404 of Figure 10 can teach "for each object in said type set, creating an object from said data in persistent storage", as recited in claim 1.

In addition, claim 1 recites the steps of: (1) obtaining the unique object identifier corresponding to said object pointer for each object pointer in said objects; and (2) obtaining the object address corresponding to said unique object identifier and setting each of said object pointers to said corresponding object addresses for each obtained unique object identifier. Greef et al. does not appear to disclose or suggest these steps.

The Examiner cites to column 4, lines 59-60 for anticipating the step labeled (1) above. Column 4, lines 52-61 of Greef et al. states:

> When an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. This is done by invoking an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier. The Data Store returns a Data Cursor with the object's contents. A smart pointer is then created and the smart pointer asks the Class that corresponds to the object's class identifier, to create an object of that class [Emphasis Added].

As can be seen, this passage of Greef et al. merely states that when loading on object, a smart pointer is created which then asks the Class that corresponds to the object's class identifier to create an object of that class [Emphasis Added]. However, this clearly does not disclose or suggest obtaining a unique object identifier that corresponds to the object pointer for each object pointer in the objects, as recited in claim 1. In fact, the above passage of Greef et al. appears to relate to a method for creating an object during an object load, while the step labeled (1) above relates to a method for resolving object pointers within an object during an object load.

Next, the Examiner cited to column 4, lines 59-61 of Greef et al. as disclosing the step labeled (2) above. The passage at column 5, lines 59-61 of Greef et al. has already been reproduced above. The cited passage of Greef et al. clearly does not disclose obtaining the object address corresponding to the unique object identifier and setting each of said object pointers to said corresponding object addresses for each obtained unique object identifier, as recited in claim 1. As noted above, the cited passage of Greef et al. appears to relate to a method of creating an object during an object load, while the step labeled (2) relates to a method for resolving object pointers within an object during an object load.

In addition to the foregoing, Greef et al. state:

> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added].

(Greef et al., column 4, lines 32-41). As such, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to obtain the object address corresponding to the unique object identifier and setting each of said object pointers to said corresponding object addresses for each obtained unique object identifier, as recited in claim 1.

On page 19 of the Final Office Action, the Examiner states that Figure 4 of Greef et al. shows objects that are to be stored persistently. The Examiner also states that "Persistent-Object A has a reference to an instance of Persistent-Object B". The Examiner then states that because these objects are to be persistently stored, and ultimately restored, then the relationship between the objects must be preserved through storing/restoring of the objects. As such, the Examiner concludes, when Greef et al. creates the objects from persistent storage, the pointers and reference of that object must be created as well; otherwise, the object would not be usable as external references would not be valid.

Applicant would like to remind the Examiner that claim 1 is a method claim. There are many methods for create objects from persistent storage. Claim 1 recites only one such method. In order to reject claim 1 under 35 U.S.C. § 102(e), the Examiner must show that Greef et al. discloses each and every element of claim 1. In Applicants view, the Examiner has failed to do so.

It appears that the Examiner is attempting to rely on inherency. However, as noted in MPEP § 1212(IV):

> The fact that a certain result or characteristic may occur or be present in the prior art is not sufficient to establish the inherency of that result or characteristic. *In re Rijckaert*, 9 F.3d 1531, 1534, 28 USPQ2d 1955, 1957 (Fed.

> Cir. 1993) (reversed rejection because inherency was based on what would result due to optimization of conditions, not what was necessarily present in the prior art); *In re Oelrich*, 666 F.2d 578, 581-82, 212 USPQ 323, 326 (CCPA 1981). "To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.'

(Emphasis Added). As noted above, there would be many ways to create objects from persistent storage. Claim 1 recites only one such method. Nothing in Greef et al. teaches many of the steps of claim 1, including the steps of: (1) obtaining the unique object identifier corresponding to said object pointer for each object pointer in said objects; and (2) obtaining the object address corresponding to said unique object identifier and setting each of said object pointers to said corresponding object addresses for each obtained unique object identifier, as recited in claim 1. In fact, and as detailed above, it appears Greef et al. use a different approach, namely, the use of smart pointers. In view of the foregoing, claim 1 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claims 2-3 are also believed to be clearly patentable over Greef et al.

Specifically with respect to claim 2, the Examiner states that Greef et al. disclose objects created in a first pass and that the objects' pointers values are set in a second pass subsequent to the first pass (citing Figure 10, items 404 and 407 of Greef et al., and the corresponding sections of the description). In reviewing Greef et al., item 404 of Figure 10 appears to relate to creating an object of a class, and item 407 appears to relate to initializing the object. Nothing in item 404, item 407 or the corresponding disclosure of Figure 10 of Greef et al. appears to disclose setting

each of said object pointers to the corresponding object addresses during a second pass, as recited in claim 2. In fact, and as noted above, Greef et al. state:

> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added].

(Greef et al., column 4, lines 32-41). As can be seen, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to set "each of said object pointers to said corresponding object addresses during a second pass", as recited in claim 2. For these additional reasons, dependent claim 2 is believed to be clearly patentable over Greef et al.

Now turning specifically to claim 3, which recites:

> 3. (Original) A method as recited in claim 1, wherein for each of said object pointers in each of said objects:
> attempting to obtain said object addresses, and setting said object pointers equal to said object address if said pointed to objects exist, otherwise deferring said setting object pointer step until said object exists.

(Emphasis Added). In contrast to claim 3, Greef et al. state:

> Note that in C++ both constant and reference numbers must be initialized in the constructor of the class. They cannot be given values at any other time.

(Greef et al., column 5, lines 20-23). As such, Applicant does not see how Greef et al. can disclose the step recited in claim 3. Like above, the Examiner appears to be relying on inherency. However, as detailed above, to establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present in the thing described in the

reference, and that it would be so recognized by persons of ordinary skill. Also, Inherency may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient. In view of the foregoing, claim 3 is believed to be clearly patentable over Greef et al.

Turning now to claim 4, which recites:

> 4. (Previously Presented) A method for writing a plurality of objects in non-persistent storage to persistent storage, the objects having pointers to objects, unique object identifiers, and object types as attributes, the method comprising the steps of:
> providing one or more common interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage;
> grouping together said objects into type sets, wherein each of said objects in each of said type sets have the same type, wherein each of said type sets have a set population equal to a total number of objects inhabiting said type set;
> counting each of said type sets and arriving at a total number of sets;
> converting each of said objects to a persistable form including obtaining a persistable form for each of said pointers to objects by obtaining a unique object identifier corresponding to each of said pointers to objects;
> writing said total number of type sets to persistent storage; and
> writing each of said type sets to persistent storage.

Greef et al. do not appear to disclose or suggest providing one or more common interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage. In fact, the persistent object classes of Greef et al. do not appear to be interfaced based as all.

In one illustrative embodiments of the present invention, the one or more common interfaces may include and be defined for the IPersistFile 20, IUnknown 28, and IPersistStream 36 classes, which are pre-defined by the COM (Component Object Model) specification (see, Specification, page 7, lines 12-22; Figure 1). This illustrative embodiment may be used to, for

example, extend component object model (COM) objects to support persistence. This is something that Greef et al. do not teach or contemplate.

In addition, Greef et al. do not appear to disclose the step of grouping together the objects into type sets, wherein each of said objects in each of said type sets have the same type. On page 5 of the Final Office Action, the Examiner cites to column 2, lines 16-17 of Greef et al. which states "Objects can be stored individually or grouped with other objects". As noted above, in order to reject claim 4 under 35 U.S.C. § 102(e), the Examiner must show that Greef et al. discloses each and every element of claim 1. The cited passage of Greef et al. clearly does not disclose or suggest grouping together said objects into type sets, wherein each of said objects in each of said type sets have the same type, as recited in claim 4. It only states that the objects can be stored individually, or grouped with other objects in some undisclosed manner.

With respect to the "counting" step of claim 4, the Examiner cites to the "class count field" in column 6, line 45 of Greef et al. However, as noted at column 4, lines 39-40 of Greef et al., the "class count field" referred to by the Examiner is a count of the classes in the object's class hierarchy. For example, and referring to Figure 4 of Greef et al., Persistent-Object-A will have two classes, Persistent-Object-B will have two classes, and Persistent-Object-C will have three classes (Greef et al., column 5, lines 40-44). As noted above, claim 4 recites the steps of:

> grouping together said objects into type sets, wherein each of said objects in each of said type sets have the same type, wherein each of said type sets have a set population equal to a total number of objects inhabiting said type set;
> counting each of said type sets and arriving at a total number of sets;

Applicant does not believe it can be readily argued that "a count of the classes in the object's class hierarchy" can correspond to "counting each of said type sets", after said objects are

grouped into type sets, wherein each of said objects in each of said type sets have the same type, and wherein each of said type sets have a set population equal to a total number of objects inhabiting said type set", as recited in claim 4. Greef et al. also do not appear to disclose the converting and writing steps of claim 4. Thus, for these and other reasons, claim 4 is believed to be clearly patentable over Greef et al.

Turning now to claim 5, which recites:

> 5. (Previously Presented) A method for storing and restoring user objects to persistent storage, the method comprising the steps of:
> providing one or more common interfaces that are used by the user objects to store and restore the objects to/from persistent storage [Emphasis Added];
> creating a persistence controller object for managing the persistence of the user objects, the persistence controller object being derived from at least one of the common interfaces;
> providing a plurality of user defined classes, the classes derived from a common object base class;
> creating a plurality of instances of user objects belonging to the user defined classes;
> providing a stream-in method and a stream-out method for each of the user defined classes;
> registering each added user defined class and added user object in a registry;
> grouping the objects according to class;
> storing the grouped user objects to persistent storage using the stream-out methods;
> loading the stored objects from storage into memory using the stream-in methods; and
> registering the user objects in the registry.

As can be seen, claim 5 recites, among other things, the steps of: providing one or more common interfaces that are used by the user objects to store and restore the objects to/from persistent storage; and creating a persistence controller object for managing the persistence of the user objects, the persistence controller object being derived from at least one of the common interfaces. Thus, for the same reasons discussed above with respect to claim 4, as well as other

reasons, claim 5 is believed to be clearly patentable over Greef et al.

In addition, claim 5 recites the step of creating a persistence controller object for managing the persistence of the user objects, the persistence controller object being derived from at least one of the common interfaces. On page 6 of the Final Office Action, the Examiner states that the two primary classes for streaming objects into and out of persistence are the Data Base Cursor and the Data Store, citing column 4, lines 42-43 of Greef et al. With respect to the persistence controller object, the Examiner cites to Figure 2 and the corresponding sections of the disclosure. However, the Data Base Cursor and the Data Store objects in Figure 2 do not manage the persistence of the user objects, as recited in claim 5. Rather, it appears that the Entity Cache (see Figure 1 of Greef et al.) may manage the persistence of objects. As noted at column 4, lines 48-50 of Greef et al.:

> When you call save on an entity cache, the entity cache calls for a Data Cursor from the data store. For each dirty Smart Pointer, the Entity Cache calls the storer method with the Data Cursor as the argument.

This would suggest that the Entity Cache is in some way managing the persistence of objects, and not the Data Base Cursor and the Data Store, as the Examiner suggests. Also, there is no indication in Greef et al. that the Entity Cache is <u>derived</u> from the Data Base Cursor or the Data Store, which the Examiner asserts, correspond to the one or more common interfaces recited in claim 5.

Furthermore, and as detailed above, Greef et al. do not appear to disclose the step of "grouping the objects according to class", as well as other steps of claim 5. For these and other reasons, claim 5 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claim 6 is also believed to be clearly patentable over Greef et al.

In addition, claim 6 recites the step of: obtaining a new address of each loaded user object, and using the stored unique identifier associated with each pointer along with the new address to set each pointer value to the new address. As noted above, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to "set each pointer value to the new address", as recited in claim 6. Also, the Examiner appears to be relying on inherency, which as detailed above, is believed to be improper under these circumstances. Thus, for these additional reasons, dependent claim 6 is believed to be clearly patentable over Greef et al.

Turning now to claim 7, which recites:

> 7. (Previously Presented) An object adapted for persistent storage, the object having a smart pointer, wherein the smart pointer includes an address attribute for containing the address of an object [Emphasis Added], and an object unique identifier attribute for containing the unique identifier of an object, wherein the object smart pointer has an assignment operation which stores the address of the object being pointed to and the unique identifier of the object being pointed to, and wherein the object includes a load method for using the smart pointer unique identifier attribute to determine and load a new smart pointer address attribute after the object being pointed to is loaded from persistent storage [Emphasis Added].

On page 8 of the Final Office Action, the Examiner states that the smart pointer of Greef et al. includes an address attribute for containing the address of an object, citing column 4, lines 52-54 of Greef et al. Column 4, lines 52-57 of Greef et al. states:

> "[w]hen an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. This is done by invoking an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier".

The Examiner concludes from this that the smart pointers of Greef et al. must inherently contain an address of the object that it points to.

Applicant must respectfully disagree. Greef et al. also state:

> The Entity Cache comprises collections of "smart pointers" that have been created during a session. FIG. 1 shows the Entity Cache comprising a collection of new entities, a collection of dirty entities and a collection of retrieved entities. <u>Each entity in the system has a unique object identifier and a class identifier as shown in the Entity Class depicted in FIG. 1</u> [Emphasis Added]. The unique object identifier is generated by the Entity ID Generator shown in FIG. 1. <u>When the object is referenced with the entity identifier, the entity cache swivels the identifier to a smart pointer</u> [Emphasis Added]. If a smart pointer already exists in the entity cache, it returns the smart pointer. If the smart pointer does not exist in the entity cache, the object is loaded from the nonvolatile memory and the smart pointer is returned to the entity in the entity cache. All persistent objects have an entity identifier.
>
> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. <u>All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves</u> [Emphasis Added].

(Greef et al., column 4, lines 17-41). First, and as noted above, Greef et al. state that "[a]ll objects manipulate smart pointers. Objects have lists of smart pointers, <u>not pointers to themselves</u>." (Greef et al., column 4, lines 49-41). This suggests that the smart pointers of Greef et al. do not include both an object unique identifier attribute for containing the unique identifier of an object <u>AND</u> an address attribute for containing the address of an object, as recited in claim 7. Instead, the smart pointers of Greef et al. appear to only include a unique identifier, which is generated by the Entity Class depicted in Figure 1 (Greef et al., column 4, lines 23-24). The Entity Cache, Data Store or some other object may generate an actual address, but it is not clear from the Greef et al. disclosure.

Greef et al. also state that <u>when the object is referenced with the entity identifier, the</u>

entity cache swivels the identifier to a smart pointer. This also indicates that the smart pointers of Greef et al. only include a unique identifier, and that the entity cache is used to swivel the identifier to a smart pointer.

The Examiner specifically points to column 4, lines 52-57 of Greef et al. However, this passage states that when an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. However, this passage also states that this is done by invoking an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier. Thus, in this example, the smart pointer appears to store and deliver an object's unique identifier to the Data Store, and it is the Data Store, and not the smart pointer, that finds the object in the non-volatile memory.

In view of the foregoing, Applicant does not believe it can readily be argued that the smart pointers of Greef et al. inherently include both an object unique identifier attribute for containing the unique identifier of an object AND an address attribute for containing the address of an object, as recited in claim 7. As noted in MPEP § 2112(IV), the fact that a certain result or characteristic may occur or be present in the prior art is not sufficient to establish the inherency of that result or characteristic. To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present [Emphasis Added] in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency may not be established by probabilities or possibilities.

In addition to the foregoing, Greef et al. do not appear to suggest an object adapted for

persistent storage that itself <u>includes a load method</u> for using the smart pointer unique identifier attribute to determine and load a new smart pointer <u>address attribute</u> after the object being pointed to is loaded from persistent storage, as recited in claim 7. As noted above, it does not appear that the smart pointers of Greef et al. even have an address attribute. In addition, Greef et al. indicate that it is the <u>Data Store</u> that finds an object in the nonvolatile memory when it is provided with the object's unique identifier, and not a load method of the object itself.

If the Examiner elects to maintain this rejection, Applicant respectfully requests that the Examiner specifically point out where Greef et al. disclose: <u>an object</u> adapted for persistent storage that itself includes <u>a load method</u> for using the smart pointer unique identifier attribute to determine and load a new smart pointer <u>address attribute</u> after the object being pointed to is loaded from persistent storage, as recited in claim 7.

For these and other reasons, claim 7 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claim 8 is also believed to be clearly patentable over Greef et al. Claim 9 has been canceled without prejudice.

Specifically with respect to claim 8, Greef et al. do not appear to suggest an object that is adapted for persistent storage, wherein the object itself includes a stream-out method for streaming out the smart pointer address attribute and unique identifier attribute to persistent storage. In Greef et al., the two primary classes for streaming objects into and out of persistent storage are the Data Base Cursor and the Data Store (see, Greef et al., column 4, lines 42-44). Nothing in the discussion of Figures 7-8 of Greef et al., which relates to storing objects to non-volatile memory, appears to disclose or suggest an object that is adapted for persistent storage, wherein the object itself includes a stream-out method for streaming out the smart pointer

address attribute and unique identifier attribute to persistent storage. If the Examiner elects to maintain this rejection, Applicant respectfully requests that the Examiner specifically point out where in Greef et al. each and every element of claim 8 can be found.

Turning now to claim 10, which recites:

> 10. (Previously Presented) A method for writing computer objects to persistent storage, the method including the steps of:
>
> providing a plurality of software objects to be stored, wherein the objects are instantiations of at least one class to be storable, wherein the storage is in persistent storage, wherein each of the classes has a unique class ID, and wherein each of the objects has a unique object ID;
>
> providing smart pointers for at least some objects, wherein the smart pointers include an address portion to contain the address of the object being pointed to and an object identifier portion to contain an object identifier of the object being pointed to;
>
> providing a persistent object controller for controlling the lifecycle of objects to be saved to persistent storage and loaded from persistent storage;
>
> providing a Persistent Object Registry for maintaining a database of objects to be saved to persistent storage, wherein the Persistent Object Registry is in communication with the persistent controller object;
>
> providing a first save method to save all objects in the Persistent Object Registry to persistent storage;
>
> providing a second save method for saving the attributes of each class having objects to be saved to persistent storage, wherein the second save method is called by the first save method;
>
> providing a first load method for loading all objects saved in a file in persistent storage;
>
> providing a second load method for loading the attributes of each class having objects to be loaded from persistent storage, wherein the second load method is called by the first load method;
>
> registering the objects to be saved with the Persistent Object Registry using the persistent object controller, including storing the class ID and object ID of the objects to be saved;
>
> writing the objects to be saved to persistent storage using the first save method and second save method;
>
> reading the objects stored from persistent storage using the first load method and second load method;
>
> registering the objects loaded into the Persistent Object Registry; and
>
> resolving the smart pointer object address attributes by using the object ID attribute value to search the Persistent Object Registry to retrieve the current

address of the object being pointed to.

As noted above, Greef et al. do not appear to disclose or suggest smart pointers that include an address portion to contain the address of the object being pointed to AND an object identifier portion to contain an object identifier of the object being pointed to. In addition, Greef et al. do not appear to disclose or suggest the step of resolving the smart pointer object address attributes by using the object ID attribute value to search the Persistent Object Registry to retrieve the current address of the object being pointed to.

In addition, Greef et al. do not appear to disclose or suggest many of the other elements of claim 10. For example, Greef et al. do not appear to suggest providing a first save method to save all objects in the Persistent Object Registry to persistent storage AND a second save method for saving the attributes of each class having objects to be saved to persistent storage, wherein the second save method is called by the first save method. On page 10 of the Final Office Action, the Examiner cites to item 101 of Figure 7 of Greef et al. as corresponding to the first save method, and item 102 of Figure 7 as corresponding to the second save method. With respect to items 101 and 102, Greef et al. state:

> A new DataCursor object and the object identity for the object to be stored are passed as parameters of this method 101. The data store then invokes the Storer method on the object to be stored 102. The next step 103 (designated by the letter "A") has the object (which is an entity) fill the DataCursor's buffer with class member data, class identifiers and class data offsets. Once we have a buffer of information representing the record of the object as shown in FIG. 6, the particular implementation of the data store saves it to permanent storage using the object identifier as the unique key 104. The process is then stopped 105.

(Greef et al., column 5, lines 53-64). As can be seen, item 101 represents the step of passing a new DataCursor object and the object identity for the object to be stored. This is clearly not

equivalent to "providing a first save method to save all objects in the Persistent Object Registry to persistent storage", as recited in claim 10. Item 102 of Greef et al. represents the step of invoking the Storer method on the object to be stored. This is clearly not equivalent to providing a second save method for saving the attributes of each class having objects to be saved to persistent storage. Finally, there is no indication that the first save method calls the second save method.

Likewise, Greef et al. do not appear to suggest providing a first load method for loading all objects saved in a file in persistent storage, and providing a second load method for loading the attributes of each class having objects to be loaded from persistent storage, wherein the second load method is called by the first load method. On page 10 of the Final Office Action, the Examiner cites to item 301 of Figure 9 of Greef et al. as corresponding to the first load method and item 302 of Figure 9 as corresponding to the second load method. With respect to Figure 9, Greef et al. state:

> Retrieving objects from the permanent storage system shown in Flowchart FIG. 9 depicts the retrieving of objects that are stored in the nonvolatile storage. FIG. 9 at 300 starts the process, when we have the identification of the object to be retrieved from non-volatile store. The DataStore for a particular permanent storage implementation is invoked with the object identifier and the data cursor. This in turn invokes the implementation dependent persistent manager's fetch method 302 and the block of data related to the object identifier key is retrieved 303 from the permanent store in a data cursor buffer. The step designated by the letter "B" is then performed where the object is created in the volatile store 304. The retrieval method is then completed and the process stops at 305.

From this, Applicant does not believe it can readily be argued that Greef et al. discloses providing a first load method for loading all objects saved in a file in persistent storage, and providing a second load method for loading the attributes of each class having objects to be

loaded from persistent storage, wherein the second load method is called by the first load method, as recited in claim 10. For the reasons given above, as well as other reasons, claim 10 is believed to be clearly patentable over Greef et al. For similar and other reasons, claim 11 is also believed to be clearly patentable over Greef et al.

On page 21 of the Final Office Action, the Examiner does not address many of the above remarks, even though these remarks were provided in Applicant's previous response. Applicant respectfully requests that the Examiner specifically point out where in Greef et al. each and every element of claim 10 can be found.

Turning now to claim 12, which recites:

12. (Previously Presented) A method for managing persistent object lifecycles, the method comprising the steps of:

providing for each object a unique object identifier attribute, an object type attribute, and an object address;

providing an object registry object for maintaining a correspondence between said unique object identifier attribute, said object address, and said object type attributes;

creating a first object having a first object type, a first object address, and a first unique object identifier, and storing said first unique object identifier, address, and type in said object registry;

creating a second object having a second object type, a second object address, and a second unique object identifier, and storing said second unique object identifier, address, and type in said object registry, said second object having a pointer attribute set equal to said first object address;

providing said second object pointer attribute to said object registry and obtaining said first object unique identifier corresponding to said second object pointer attribute in return;

writing said second object to persistent storage as second object data, and writing said first object unique identifier corresponding to said second object pointer attribute to persistent storage, such that said written first object unique identifier is associated with said second object pointer attribute in persistent storage;

deleting said second object from non-persistent storage;

reading said second object data from persistent storage and creating said second object having said second object type;

reading said first object unique identifier associated with said second object data from persistent storage;

providing said object registry with said first object unique identifier and obtaining said first object address in return; and

setting said second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object.

As noted above, Greef et al. does not disclose or suggest the step of setting said second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object, as recited in claim 12. Instead, Greef et al. state that "[a]ll objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added]." (Greef et al., column 4, lines 49-41). Halter does not appear to add anything to Greef et al. in this regard.

In addition, claim 12 recites "providing an object registry object for maintaining a correspondence between said unique object identifier attribute, said object address, and said object type attributes". On page 16 of the Final Office Action, the Examiner appears to be suggesting that a "database" of Greef et al. corresponds to the object registry object of claim 12, and that the Data Store is an object of the database registry. First off, it is not clear how the Data Store can be an object of a database, and there is certainly no indication in Greef et al. that the Data Store is an object of a database (see Fig. 2 of Greef et al.). In addition, assuming the "database" of Greef et al. corresponds to the object registry of claim 12, there is no indication in Greef et al. that the "database" is an object. Claim 12 recites an "object registry object", and that the "object registry object" maintains a correspondence between said unique object identifier attribute, said object address, and said object type attributes.

Claim 12 also recites "creating a first object having a first object type, a first object address, and a first unique object identifier, and storing said first unique object identifier, address, and type in said object registry". Again, Greef et al. do not appear to disclose or suggest an "object registry object", and therefore Greef et al. cannot suggest "creating a first object having a first object type, a first object address, and a first unique object identifier, and storing said first unique object identifier, address, and type in said object registry", as claimed.

Claim 12 also recites "creating a second object having a second object type, a second object address, and a second unique object identifier, and storing said second unique object identifier, address, and type in said object registry, said second object having a pointer attribute set equal to said first object address". On page 16 of the Final Office Action, the Examiner cites to Figure 4 of Greef et al., and the corresponding section of the disclosure. The Examiner notes that Persistent-Object-C in Figure 4 contains a pointer to Persistent-Object-B.

First, Greef et al. do not appear to disclose or suggest an "object registry object", and therefore Greef et al. cannot suggest "creating a second object having a second object type, a second object address, and a second unique object identifier, and storing said second unique object identifier, address, and type in said object registry. Second, Greef et al. clearly do not suggest a second object, wherein the second object has a pointer attribute set equal to said first object address. As detailed above, in Greef et al., each entity has a corresponding smart pointer. It is a sub for a real object in memory. It has no data members and it overloads the pointer and de-reference operators. All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves. (see, Greef et al., column 4, lines 32-41). Therefore, Greef et al. would appear to actually teach away from providing a second object that has a pointer

attribute that is set equal to said first object address, as claimed.

Claim 12 also recites "providing said second object pointer attribute to said object registry and obtaining said first object unique identifier corresponding to said second object pointer attribute in return". On page 16 of the Final Office Action, the Examiner cites to column 4, lines 58-60 of Greef et al, which states "[a] smart pointer is created and the smart pointer asks the Class that corresponds to the object's class identifier, to create an object of that class." Applicant does not see how this discloses or suggests "providing said second object pointer attribute to said object registry and obtaining said first object unique identifier corresponding to said second object pointer attribute in return". Further clarification by the Examiner is requested.

Claim 12 also recites "writing said second object to persistent storage as second object data, and writing said first object unique identifier corresponding to said second object pointer attribute to persistent storage, such that said written first object unique identifier is associated with said second object pointer attribute in persistent storage". On page 16 of the Final Office Action, the Examiner cites to column 6, lines 2-6 which states "[i]f the object is not the top of the class hierarchy...then the storage method on its super classes are invoked as noted by the query...". Applicant does not see how this discloses or suggest the step of "writing said second object to persistent storage as second object data, and writing said first object unique identifier corresponding to said second object pointer attribute to persistent storage, such that said written first object unique identifier is associated with said second object pointer attribute in persistent storage". Further clarification by the Examiner is requested.

Claim 12 also recites the steps of "reading said second object data from persistent storage and creating said second object having said second object type", "reading said first object unique

identifier associated with said second object data from persistent storage, "providing said object registry with said first object unique identifier and obtaining said first object address in return", and setting said second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object." As detailed above, Greef et al. clearly do not suggest providing a object registry with a first object unique identifier and obtaining the first object address in return, and setting the second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object, as recited in claim 12. In Greef et al., each entity has a corresponding smart pointer, and all objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves. (see, Greef et al., column 4, lines 32-41). Therefore, Greef et al. would appear to actually teach away from providing a object registry with a first object unique identifier and obtaining the first object address in return, and setting the second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object, as recited in claim 12. Halter does not appear to add anything to Greef et al. in this regard. Thus, for these and other reasons, claim 12 is believed to be clearly patentable over Greef et al. in view of Halter.

Now turning to claim 13, which recites:

> 13. (Previously Presented) A method for writing a plurality of objects in non-persistent storage to persistent storage, the method comprising the steps of:
> providing one or more common object interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage;
> each of the common object interfaces having a corresponding common object class;

providing a Persistent Object Registry for maintaining a listing of objects to be saved to persistent storage, wherein the Persistent Object Registry is in communication with a persistent controller object, and the persistent controller object is derived from one or more of the common object classes; and

providing a first save method to save all objects in the Persistent Object Registry to persistent storage.

As can be seen, claim 13 recites, among other things, the steps of: providing one or more common object interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage; providing a Persistent Object Registry for maintaining a listing of objects to be saved to persistent storage, wherein the Persistent Object Registry is in communication with a persistent controller object, and the persistent controller object is derived from one or more of the common object classes; and providing a first save method to save all objects in the Persistent Object Registry to persistent storage. Thus, for the same reasons discussed above with respect to claims 4 and 5, as well as other reasons, claim 13 is believed to be clearly patentable over Greef et al.

In addition, claim 13 recites that the persistence controller object is derived from at least one of the common interfaces. On page 11 of the Final Office Action, the Examiner states that the two primary classes for streaming objects into and out of persistence is the Data Base Cursor and the Data Store, citing column 4, lines 42-43 of Greef et al. With respect to the persistence controller object, the Examiner cites to Figure 2, and column 4, lines 47-48 of Greef et al. However, the Data Base Cursor and the Data Store objects in Figure 2 do not appear to be a persistence controller object at all. Rather, it appears that the Entity Cache (see Figure 1 of Greef et al.) may manage the persistence of objects, but it is not entirely clear. As noted at column 4, lines 48-50 of Greef et al.:

> When you call save on an entity cache, the entity cache calls for a Data Cursor from the data store. For each dirty Smart Pointer, the Entity Cache calls the storer method with the Data Cursor as the argument.

This would suggest that the Entity Cache is in some way managing the persistence of objects, and not the Data Base Cursor and the Data Store, as the Examiner appears to be suggesting. Also, there is no indication in Greef et al. that the Entity Cache is derived from the Data Base Cursor or the Data Store, which the Examiner asserts, correspond to the one or more common interfaces recited in claim 13.

Furthermore, claim 13 recites the step of "providing a first save method to save all objects in the Persistent Object Registry to persistent storage". On page 12 of the Final Office Action, the Examiner cites to item 101 of Figure 7 of Greef et al. as corresponding to the first save method. With respect to items 101 and 102, Greef et al. state:

> A new DataCursor object and the object identity for the object to be stored are passed as parameters of this method 101. The data store then invokes the Storer method on the object to be stored 102. The next step 103 (designated by the letter "A") has the object (which is an entity) fill the DataCursor's buffer with class member data, class identifiers and class data offsets. Once we have a buffer of information representing the record of the object as shown in FIG. 6, the particular implementation of the data store saves it to permanent storage using the object identifier as the unique key 104. The process is then stopped 105.

(Greef et al., column 5, lines 53-64). As can be seen, item 101 represents the step of passing a new DataCursor object and the object identity for the object to be stored. This is clearly not equivalent to ""providing a first save method to save all objects in the Persistent Object Registry to persistent storage", as recited in claim 13. For these and other reasons, claim 13 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claims 14-18 are also believed to be clearly patentable over Greef et al.

Now turning to claim 19, which recites:

> 19. (Previously Presented) A method for reading a plurality of objects from persistent storage to non-persistent storage, the method comprising the steps of:
>
> providing one or more common object interfaces that are used by each of the plurality of objects to load the objects from persistent storage to non-persistent storage;
>
> each of the common object interfaces having a corresponding common object class;
>
> providing a Persistent Object Registry for maintaining a listing of objects that are loaded from persistent storage, wherein the Persistent Object Registry is in communication with a persistent controller object, and the persistent controller object is derived from one or more of the common object classes; and
>
> providing a load method to load objects in the Persistent Object Registry to non-persistent storage.

For similar reasons to those discussed above with respect to claim 13, as well as other reasons, claim 19 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claims 20-24 are also believed to be clearly patentable over Greef et al.

Finally, Applicant has added newly presented claim 25, which is also believed to be clearly patentable over the prior art of record.

In view of the foregoing, Applicant believes that all pending claims 1-8, 10-25 are in condition for allowance. Reexamination and reconsideration are respectfully requested. If the Examiner believes it would be beneficial to discuss the application or its examination in any way, please call the undersigned attorney at (612) 359-9348.

Respectfully submitted,

Dated: August 31, 2005

Brian N. Tufte, Reg. No. 38,638
CROMPTON, SEAGER & TUFTE, LLC
1221 Nicollet Avenue, Suite 800
Minneapolis, MN 55403-2402
Telephone: (612) 677-9050
Facsimile: (612) 359-9349